RESEARCH PAPER

KnowID: An Architecture for Efficient Knowledge-Driven Information and Data Access

Pablo Rubén Fillottrani[1,2] & C. Maria Keet[3†]

[1]Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Buenos Aires 8000, Argentina

[2]Comisión de Investigaciones Científicas, Provincia de Buenos Aires, Argentina

[3]Department of Computer Science, University of Cape Town, Rondebosch 7700, South Africa

Keywords: Extended entity-relationship diagrams; Abstract relational model; Ontology-based data access

Citation: P.R. Fillottrani & C.M. Keet. KnowID: An architecture for efficient knowledge-driven information and data access. Data Intelligence 2(2020), 487–512. doi: 10.1162/dint_a_00060

Received: October 8, 2019; Revised: April 24, 2020; Accepted: April 26, 2020

ABSTRACT

Modern information systems require the orchestration of ontologies, conceptual data modeling techniques, and efficient data management so as to provide a means for better informed decision-making and to keep up with new requirements in organizational needs. A major question in delivering such systems, is which components to design and put together to make up the required "knowledge to data" pipeline, as each component and process has trade-offs. In this paper, we introduce a new knowledge-to-data architecture, KnowID. It pulls together both recently proposed components and we add novel transformation rules between Enhanced Entity-Relationship (EER) and the Abstract Relational Model to complete the pipeline. KnowID's main distinctive architectural features, compared to other ontology-based data access approaches, are that runtime use can avail of the closed world assumption commonly used in information systems and of full SQL augmented with path queries.

1. INTRODUCTION

Traditional data management comprises a sequence from requirements to conceptual data model, transforming it into a relational model, and from there creating a physical database schema, at each stage leaving behind the artefact produced in the preceding step. As valuable information is represented in the models early in the pipeline, ideas for trying to reuse the outputs of the first three stages have been proposed

† Corresponding author: C. Maria Keet (Email: mkeet@cs.uct.ac.za, ORCID: 0000-0002-8281-0853).

over the years, such as query-by-diagram with a Unified Modeling Language (UML)-like notation [1] and conceptual queries with Object-Role Modeling (ORM) [2] in the mid-1990s. From the mid-2000s, theory, techniques, and tools started to advance to reuse the conceptual model *at runtime* in conjunction with large data stores to create a "knowledge to data" pipeline. This combination, or pipeline, is named with various terms, including ontology-based data access and (virtual) knowledge graphs [3]. Let us illustrate a key advantage of such a "pipeline" with an example about an information request about customers.

Example 1. *A member of the Marketing department of some company, say, Juan, would like to ask the company's information systems about customers. If there are no canned queries and Juan cannot write Structured Query Language (SQL) or does not know in which database(s) the customers are stored, then he either has to explore the data himself or ask a database administrator for help. Both options cost time, and time is money. It has been reported that data scientists spend at least 80% of their time on data discovery and integration [4].*

Imagine now there is one model representing what data are stored in the systems, possibly graphically displayed. Then Juan himself can find what he can query, reducing the effort of that otherwise time-consuming data discovery task. With an "intelligent" system, he does not have to know where and how exactly that data are stored in which database, just that it is somewhere, and simply pose the query, say, "list all customers". Here, the scenario diverges on a second aspect, notably cf. query-by-diagram approaches, for in the knowledge-to-data pipeline, the model is used at runtime to answer the query. To see, this, let us assume that `Joanne` *is an* IndividualCustomer *that is a* Customer*; formally, we have* `IndividualCustomer` ⟨`Joanne`⟩ *in the database (i.e., a table named* `IndividualCustomer` *that has a row with* `Joanne`*) and* IndividualCustomer ⊑ Customer *in the knowledge layer (e.g., a Web Ontology Language (OWL) ontology). When Juan fires the query "list all customers", he will observe different results:*

- *Default relational database installation: it returns {}, i.e., the answer is empty, as there are no explicitly declared instances of* `Customer` *in the database.*
- *The "knowledge to data" pipeline (with reasoner): it returns* {`Joanne`}*, thanks to the inference from the subsumption that inferred* `Customer` ⟨`Joanne`⟩.

An end-user, such as Juan, would expect {`Joanne`} *as answer, but we only obtain that with the second option, where the human intelligence in understanding the situation is in some way woven into the system to obtain that behavior from the system.*

The relatively most popular approach to build such intelligence into the system is called Ontology-Based Data Access [5, 6]. It uses the open world assumption (OWA) rather than the closed world assumption (CWA) of databases that endusers are more familiar with[①], contains a computational costly mapping layer to link the knowledge to the data, and supports only a fragment of full SQL (unions of conjunctive queries).

[①] e.g., with data {⟨`Joanne`, `PhD`⟩, ⟨`John`, `MSc`⟩} and a query "who doesn't have a PhD?", the answer under OWA would be {} and under CWA, it would be {`John`} because absence is treated as negation.

Another option is to store everything in a database, but it has typically only limited, or no, support for automated reasoning, and if so, it is implemented with carefully crafted stored procedures and triggers (e.g., [7, 8]). Alternatives that address these issues are being looked into. A recent alternative proposed to extend the relational model into an "Abstract Relational Model" (ARM) with special object identifiers and a strict extension to SQL for path queries (SQLP) [9]. It addresses a typical database user's expectation of CWA and supports full SQL. It avoids the computationally costly mapping layer of ontology-based data access by means of computationally cheap transformations. However, this ARM+SQLP option falls short on the knowledge (i.e., ontology or conceptual data model) layer in the knowledge-to-data pipeline in that it does not have that knowledge component.

We aim to extend this partial knowledge-to-data pipeline of ARM+SQLP into the knowledge layer by adding a conceptual data model or application ontology and relevant related model management features, in such a way that the runtime use can remain within the closed world assumption and users still will be able to use full SQL. This extension is worked out in detail for the ARM to/from Enhanced Entity-Relationship (EER) transformation by means of a set of rules, which are different from the regular EER-to/from-Relational Model (RM) transformations due to several peculiarities of ARM and the richer set of constraints that can be represented in ARM *cf*. the RM. The generalization from that can then avail of other extant recent research results to complete a generic knowledge-to-data architecture, which we call KnowID: **Know**ledge-driven **I**nformation and **D**ata access. KnowID's functionality is thus similar to ontology-based data access systems: it allows for queries to be posed at the knowledge layer—i.e., supporting a user in *what* to query, without the labor-intensive figuring out of the *how and where*—that will be evaluated over an "intelligent" database that makes use of the knowledge represented in the conceptual data model or ontology. Architecturally, a distinct practical advantage is that it achieves this through a series of automated transformations, rather than (manual or automated) specifications of non-trivial mappings in a separate mapping layer. A further practical advantage is the support for path queries, which has been shown in user experiments to make query formulation faster with at least the same level of accuracy or fewer errors [10, 11].

The remainder of the paper is structured as follows. Preliminaries and related work is presented first in Section 2, which presents an overview of the four principal knowledge-to-data approaches to place the architecture in context and a summary of relevant key components of ARM. This is followed by design considerations of the extension in Section 3, and the novel KnowID architecture is introduced. This is followed by the new technical details of the extension, being the transformations from EER to ARM in Section 4. The genericity of the architecture is motivated and discussed briefly in Section 5, and its evaluation in Section 6. We conclude in Section 7.

2. PRELIMINARIES

Accessing data through knowledge requires some background knowledge about the various components in the pipeline, which span subfields within computer science. We therefore introduce some key terms and related work first and then proceed to an overview of one approach—ARM with SQLP—in more detail.

2.1 Key Components of a Knowledge-to-Data Pipeline

We focus on combining *intensional knowledge* ($\mathcal{K}$) represented as formalized descriptions about entity types, relationships among them, and constraints holding for them—e.g., modeled in a conceptual data model or an ontology[2]—with *large amounts of data* ($\mathcal{D}$) in order to manage that data that represent instances. In surveying the literature, we distilled four core approaches, which are summarized and illustrated with exemplars.

K@*D*–**Knowledge with Data** store $\mathcal{K}$ and $\mathcal{D}$ in one system, knowledge representation (KR)-oriented, informally: "push the envelope of the AI logic to manage some data"; e.g., put $\mathcal{K}$ in the ontology's "TBox", $\mathcal{D}$ in the "ABox" (in Description Logics terminology [14]), and store both in a single OWL file [15] for use with Semantic Web infrastructure.

K⇔*D*–**Knowledge mapping Data** store $\mathcal{K}$ in a KR-oriented way, as for *K*@*D*, but delegate $\mathcal{D}$ to external storage in a database and link the two through a mapping layer ($\mathcal{M}$) that maps terms from $\mathcal{K}$ to queries over $\mathcal{D}$. Informally, this is the approach of "use each where it's good at"; e.g., Ontology-Based Data Access (OBDA) that links an OWL file to a Relational Database Management System (RDBMS) by means of the mappings [6] for which multiple tools and use cases exist [3].

D⋈*K*–**Data transformation Knowledge** exploit data, database schemas, and the relational model layer up back to its originating conceptual model or application ontology that is formalized in a suitable logic, informally: "extend the database with some AI notions"; this option is currently incomplete, but partial theoretical foundations are described in [9, 10, 16].

D@*K*–**Data with Knowledge** store both $\mathcal{K}$ and $\mathcal{D}$ in one system, database-oriented, informally: "exploit the database technologies and push the envelope thereof"; e.g., the OntoMinD system [7] has separate tables for the knowledge and for the data, and additional triggers and stored procedures in the RDBMS to cater for automated reasoning.

The respective core architectures are depicted in a simplified way in Figure 1. There are a range of design decisions to take to construct specific architectures, such as open *vs*. closed world, the query language (e.g., SPARQL, SQL, or both, or some extension thereof), choice of mapping (if applicable), the types of queries one can pose, the expressiveness of the logic for representing the knowledge in $\mathcal{K}$ (e.g., OWL to QL, OWL to RL), and the language and storage system for $\mathcal{D}$ (e.g., SQL, RDF, Datalog). Actual instances thus also may be more elaborate with more sub-components on how such an architecture may be realized. However, their respective core approach to solving the runtime use of knowledge and data is essentially

[2] Theoretically, conceptual data models are different from ontologies, largely due to the fact that the former is meant to be application-dependent and the latter application-*in*dependent, which affects their respective contents (see [12] for detail and examples). Practically, the artefacts developed constitute more of a continuum due to the mixing of modeling styles in some ontologies that make them more like application-specific artefacts [13] as well as generalizing conceptual models into reusable enterprise models that resemble ontology design patterns. It is outside the scope of this paper to dwell on these details.

unchanged. Variants include, e.g., conceptually adding a "second TBox" in the knowledge layer in OBDA, which links the OBDA ontology (*de facto* a conceptual data model) to a domain ontology [17] or there may be an additional intermediate query language for generalizability purposes [18], yet the overall approach is still that of $K \Leftrightarrow D$.

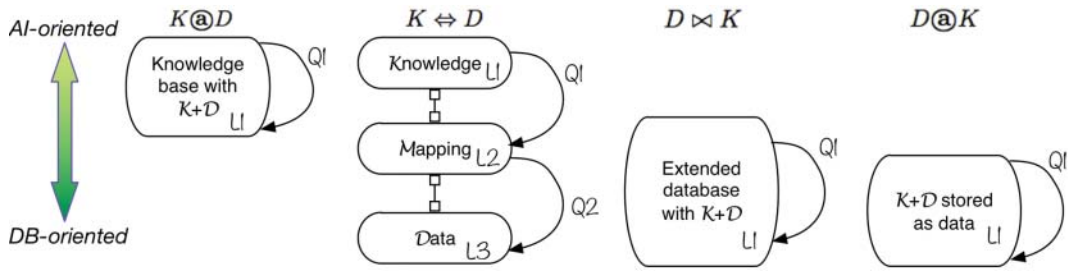

Figure 1. High-level overview of the four approaches, where Lx means (declarative) representation language and Qx the query language. $\mathcal{K}$ is shorthand for the information and knowledge-level content and $\mathcal{D}$ is shorthand for instance-level content, irrespective of the formalism and technology used.

A specific design of $K \Leftrightarrow D$ is OBDA, which has one or more particular software implementations, such as *Ontop* [5]. *Ontop* permits installations that use OWL to QL or RDFS as ontology language for $\mathcal{K}$, Quest as automated reasoner, R2RML and its own native mapping layer languages in $\mathcal{M}$, and it supports several RDBMSs for $\mathcal{D}$ including UnityJDBC, IBM Websphere, MS SQL Server, and Oracle. An example installation will then use one of the options where there is choice, such as OWL to QL and an Oracle database in the case of the Slegge system at the Statoil company [19]. A similar drill-down from the generic principal architecture to the configuration of an actual system is possible with the other architectures.

D@K systems have been proposed that exploit relational, object-relational, and object-oriented databases. They have a range of trade-offs as well, such as a comprehensive database schema to store all of the possible OWL axioms but no reasoning other than regular database querying [8] or only a fragment of OWL (e.g., *DL*-Lite$_{\mathcal{R},\sqcap}$) but with automated reasoning [7]. Their actual architectures tend to be less elaborate than the orchestrations of the systems of $K \Leftrightarrow D$ in particular; see [7] for a good overview.

The only principal approach that is incomplete in its overall design as well as implementation, is $D \bowtie K$. Therefore, we will take a closer look at it in the next section.

2.2 Comparison of Architectures

We compare the four architectures on key points that have a major impact on overall complexity of any particular practical design for such an architecture. These distinctive features are listed in Table 1 and discussed in the remainder of this section.

First, note that among options $K \Leftrightarrow D$, $D \bowtie K$, and *D@K*, it is well-known that one has to handle the automated reasoning in a special way, because it does not have the native ABox of the ontology as in *K@D*.

One can achieve this by either query rewriting (i.e., reasoning) on-the-fly or data completion, meaning to pre-compute the deductions and materialize the inferences. $\mathcal{K} \Leftrightarrow \mathcal{D}$ generally uses query rewriting (including *Ontop*) and $\mathcal{D} \bowtie \mathcal{K}$ and $\mathcal{D} @ \mathcal{K}$ (including, e.g., OntoMinD [7]) use data completion.

Second, there are other non-trivial issues that involve or affect automated reasoning, in particular:

(1) the expressiveness of the language for $\mathcal{K}$;
(2) which types of queries are supported over $\mathcal{D}$; and
(3) the class of algorithms used (e.g. tableaux *vs*. stored procedures).

Within items (1) and (2), one has to divide the options between whether that is (i) under OWA or CWA, (ii) with the unique name assumption or not, for both can affect the computational complexity reasoning problems for $\mathcal{K}$'s representation language [20], and (iii) how queries are evaluated over $\mathcal{D}$. While the OWA/CWA balance tends to be the same within an approach, they differ across the four approaches, reflecting their origins (AI or DB). These within-architecture variations for actual instances of a pipeline thus mean that there is no single computational complexity figure for each architecture.

Third, one has to appreciate the distinction between ontological principles and engineering. While the philosophical and mathematical differences between classes, instances, and values are clear, one can "play" with it in an architecture for implementation at the back-end. For instance, take customer `Joanna` again and the class IndividualCustomer:

– if `Joanna` is stored in a database, it is not an individual but a value, so if it is to be passed on to the ABox of an ontology, then that value has to be converted to an object.
– if IndividualCustomer is stored in a database, it is not a class, but a value, so it cannot have, e.g., a set of members as instances, because values do not have such instances; hence, they have to be treated in a different way from those cells that store instances-as-values.

This sort of manipulation of classes, instances, and values, also called entity recasting, is typical for $\mathcal{K} \Leftrightarrow \mathcal{D}$ and $\mathcal{D} @ \mathcal{K}$, whereas $\mathcal{K} @ \mathcal{D}$ and $\mathcal{D} \bowtie \mathcal{K}$ stay true to their respective formalisms.

Finally, as general consideration, it has been shown that domain experts' data access at, at least, the relational and also at the conceptual layer improves query accuracy and time (see [10] and references therein). Therefore, it is not uncommon that there may be a "syntactic sugar" layer on top of the principal architecture, such as the early systems of ACE for OWL with $\mathcal{K} @ \mathcal{D}$ [21] and WONDER [22] for OBDA with $\mathcal{K} \Leftrightarrow \mathcal{D}$ that provided a natural language and graphical interface to the system, respectively. More recent systems may also combine ease of user access and systems management, such as Optique [23]. We are not concerned with the quality of a presentation layer here, but note that it is a common extension for either of the four approaches.

Table 1. Summary of the distinguishing features of varying computational cost.

Feature	KⓐD	$K \Leftrightarrow D$	$D \bowtie K$	DⓐK
World	OWA	OWA+CWA	CWA	CWA
Language for $\mathcal{K}$	OWL	OWL	relational, DL	relational
Language for $\mathcal{D}$	OWL	relational	relational	relational
Query language	SPARQL	SPARQL + SQL (fragment)	SQLP	SQL
Automated reasoning	yes	yes	yes	depends on system
Reasoning w.r.t. data	no separate approach	query rewriting	data completion	data completion
Mapping layer	no	yes	no	no
Transformations	no	no	yes	yes
Entity recasting	no	yes	no	yes
Syntactic sugar	available	available	possible	possible

2.3 Preliminaries: ARM+SQLP

The Abstract Relational Model (ARM) is a generalization of the relational model (RM) which includes an abstract domain of entities `OID` used for object identifiers, and a new `self` attribute for each relation in an ARM that through an underlying theory of referring expressions remains virtual and thus does not cost one another column in a database table. Its details are described in [9, 10, 16]. This seemingly simple extension allows for various other constructs beyond primary and foreign keys, such as declaring disjointness constraints, explicit inheritance, and path functional dependencies. In short:

Definition 1 (Relations and Constraints) *Let* REL, AT, *and* CD *be sets of relation names, attribute names (including* `self`*), and* concrete domains *(data types), respectively, and let* `OID` *be an* abstract domain *of* entities (surrogates)*, disjoint from all concrete domains. An* abstract relational model schema $\sum$ *is a set of* relation declarations *of the form*[③]

$$\texttt{table}\ T(A_1\ D_1, \ldots, A_n\ D_n, \varphi_1, \ldots, \varphi_m)$$

where $T \in$ REL, $A_i \in$ AT, $D_i \in$ (CD$\cup\{$`OID`$\}$)*, and* φ_i *are* constraints *attached to relation* T*. We write* REL($\sum$) *to denote all relation names declared in ARM schema* $\sum$*.* A_i *is* abstract *if* D_i *is* `OID`*, and* concrete *otherwise, and* `self` *must be abstract for each* T*. Constraint* φ_i *has one of the following forms:*

1. (primary keys) `primary key (self)`
2. (foreign keys) `constraint` N `foreign key` (A_i) `references` T_1, *where* A_i *is an abstract attribute in the definition of* T
3. (inheritance) `isa` T_i
4. (covering constraints) `covered by` (T_1, …, T_k)
5. (disjointness constraints) `disjoint with` (T_1, …, T_k)
6. (path functional dependencies) `pathfd` (Pf_1, …, Pf_k)$\rightarrow$Pf

where each $T_i \in$ REL*. An attribute path* Pf *is either* `self` *or a dot-separated sequence of attribute names excluding* `self`*.*

[③] Observe that this essentially an abstract, implementation-independent, rendering of the syntax for SQL's CREATE TABLE command.

Primary keys and named foreign keys ("N" in item 2, above) are supported by standard SQL. Inheritance and disjointness constraints are satisfied when all (resp. no) `self`*-value occurring in T occurs as a* `self`*-value in some (resp. all)* T_i *in the inheritance (resp. disjointness) cases. A path functional dependency is satisfied when any pair of T-tuples that agree on the value of each* Pf_i *also agree on the value of* Pf.

The identifier aspects of objects with `self` and its `OID` is handled by referring expressions, which is rather involved and described in detail in [9, 10]; for the extension of the ARM+SQLP option, it suffices to know that it works. Likewise, the ARM-to-RM [9] and the RM-to-ARM [10] transformation algorithms are internal to the data layer, which does not affect the information and knowledge layer (the focus of this paper)—we may assume they are in place. The basic formalization of ARM as stated above provides the possibility to reason about constraints, as *logical consequences*. That is, for some T_i in $\sum$, $\sum \models (\varphi \in T_i)$ denotes that constraint φ for T logically follows from the constraints in $\sum$ even when not explicitly stated. A simple example is when T_1 has attribute A_1 as PK yet A_1 also appears in an FK constraint to T_2 where it is the PK as well: this implies the constraint φ = `isa` T_2 for T_1. ARM can be formalized in a Description Logic (DL) with *n*-aries, e.g., the PTIME decidable $\mathcal{CFDI}_{nc}^{\forall -}$ [24], where the problem of deciding when $\sum \models (\varphi \in T)$ holds in ARM schemata can be reduced to reasoning about logical consequence [9].

This approach has a complementary query language, called SQLP [9], which is an extension to full SQL in order to allow queries over paths, availing over those virtual identifiers in the background, and has been shown to improve querying when used with ARM [10] (this is not the focus in this paper). Note that, since SQL and the RM already operate under the CWA, so will ARM+SQLP as its extension with identifiers and paths do no affect CWA. Let us illustrate this with an example.

Example 2. *Consider a section of the sample ARM diagram that was evaluated by users by [10], which is depicted in Figure 2 on the left-hand side and that tried to emulate an RM look-and-feel (the right-hand side is alike the ARM sample diagram notation in [25] and emphasizes its graph-like structure). One of the definitions of the relations is as follows:*

```
table professor ((self OID, pnum INT, pname STRING, office STRING, department OID),
primary key (self),
constraint dept foreign key (department) references department,
pathfd (pnum) -> self,
disjoint with (course, class, department))
```

*The other definitions of the ARM relations follow the same pattern. A sample query may be "*Find the distinct names of all the professors who work at the CS department that have taught a class of a course offered by another department.*" This can be written in SQLP for the ARM as follows:*

```
SELECT distinct pname FROM professor p
    WHERE p.department.dname = 'CS'
      AND class.course.department.dcode != class.p.department.dcode
```

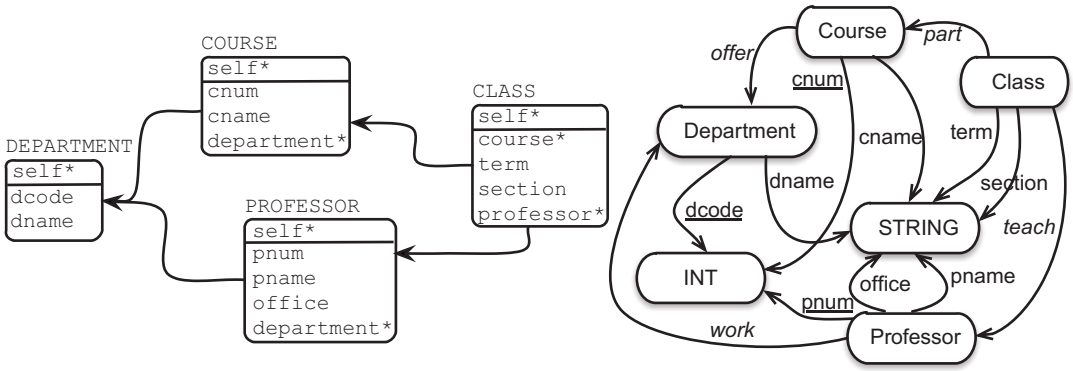

Figure 2. Two graphical renderings of a sample ARM schema, resembling a relational model graphical notation (left; source: based on a section of the diagram in [10]) and in ontology-like notation similar to the ARM notation in [25].

Recall that in an RM, all attributes must be declared to be concrete, and only primary and foreign keys constraints are allowed. In contrast, ARM includes `self` to be the primary key of every table, with object identifiers taken from an abstract domain. This means that ARM generalizes RM in two ways: first, there is no explicit choice in the representation of primary keys for each relation, we just know it exists; second, a set of declarative constraints is available to model the domain data. In this sense, ARM is, strictly, conceptual model-*like*, but it does not complete the knowledge-data column, for ARM still contains various design decisions, rather than be purely at the conceptual "what"-layer of representing information and knowledge. Those design decisions are, mainly: primary and foreign key choices and naming, data types, the choice of "a new relation for each entity type" of those entity types that are in a hierarchy in a conceptual model, and it assumes other choices, such as the FK side of a 1:1 cardinality of a relationship.

It is this last step of the additional knowledge layer that tends to make the whole pipeline theoretically and technically rather involved. We first deliberate the options for adding that knowledge layer to the ARM+SQLP approach in the next section, and subsequently add it.

3. DESIGN CONSIDERATIONS FOR ADDING A KNOWLEDGE LAYER TO ARM+SQLP

The idea of somehow having a whole knowledge-to-data pipeline or column is intuitively clear, but what does it mean with respect to its distinguishing features and requirements such that one can say "this architecture is one of those, but that one is not"? There are three key components:

- Knowledge, formally/logic-based representation of it, at the type/class-level (i.e., with entities that can be instantiated). This type-level theory should be independent of systems design aspects and only contain the "what" of the universe of discourse not the "how" components (so, e.g., no PK/FK and OO object identifiers and similar in the model);

- Structured data, which are representations of individuals (i.e., those entities that cannot be instantiated further); e.g., data as stored in a relational database or an RDF triple store (but not unstructured text documents).
- Automated reasoning support with, as a minimum, querying the data using the vocabulary elements from the knowledge layer. It should avail of the formally represented knowledge in some way (so, not just a graphical query interface for the stored data like in query-by-diagram).

Holding the ARM+SQLP option against these key components, it falls short on the knowledge layer, for its ARM specification does include design decision, as discussed in the previous section (PK/FK etc.). To complete this approach for knowledge-to-data to also include the knowledge layer, there are two broad strategies to choose from:

a. Add an ontology or conceptual data model, suitably formalized in first order logic or a fragment thereof, such as OWL or another Description Logic (DL), and link it through a mapping layer to the ARM.
b. Construct an algorithm that is similar to the forward pipeline or the reverse engineering one between conceptual models and relational models, but then adjusted for ARM rather than from/to the RM.

Both ARM and the conceptual data modeling language ORM [26] have been formalized in a member of the $\mathcal{CFD}$ family of DLs [27, 25], which makes Option *a* seem doable. However, like all DLs, the $\mathcal{CFD}$ logics operate under the open world assumption (OWA), whereas databases assume the closed world assumption (CWA), and from a theoretical and technical viewpoint linking OWA with CWA components in one system is far from trivial. Option *b* could operate fully within CWA, whilst still permitting a formalization in some DL so as to use standard automated reasoners, provided it is carefully orchestrated. This extension is sketched in Figure 3 and will be elaborated on in Sections 4 and 5. As can be seen already from the figure, it remains within CWA in the crucial stages of querying, since at query formulation only the knowledge layer vocabulary will be used and the query itself—in SQLP throughout—is fully evaluated within the CWA of the database world. Such an Option *b* is more doable theoretically than Option *a*, although it leaves to be decided how to deal with which conceptual modeling language and the transformation algorithms between that and the ARM will have to be designed. The choice of conceptual modeling language turns out not to matter, because the constraints that can be represented in the ARM can be mirrored in the popular conceptual data modeling languages, they are interchangeable either through direct 1:1 transformations or through a joint metamodel, as has been demonstrated most recently in [28], or a common core profile can be designed, as proposed in [29]. The former—those transformations—will be addressed in the next section.

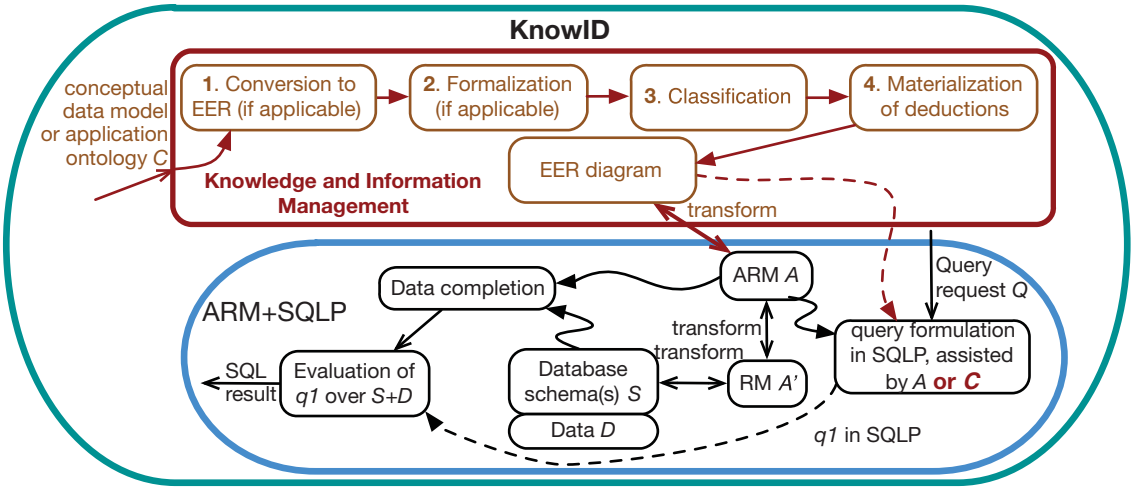

Figure 3. Extending ARM+SQLP toward KnowID, i.e., adding a knowledge layer to the architecture, into the **Know**ledge-driven **I**nformation and **D**ata access, KnowID, architecture.

4. TRANSFORMATIONS BETWEEN ARM AND CONCEPTUAL MODELS

Given that the transformations between EER and RM are well known, we will base the approach of our bi-directional algorithms on that and adjust it for ARM's peculiarities. Further, because ARM has several unique constraints that a regular RM does not have, we will present the bi-directional transformations from ARM to EER first, and then from EER to ARM. Observe that these transformations are meant to bridge the gap between the knowledge layer and the data layer, where the former has no known complete semantic formalization, i.e., including weak entity types, and as such no soundness results can be given. Also, the modeling features of both languages are diverse in several ways due to their different aims, therefore making the transformations necessarily incomplete. More details are provided in the description of the respective transformations below.

4.1 From ARM to EER diagram

The reverse engineering from ARM to EER follows the same ideas as for the usual components, notably as presented in [30], such as the classification of relations and attributes, 1:n relationships and a "dangling attribute" indicating a weak entity type. Key differences compared to the regular RM-to-EER reverse engineering algorithms can be observed in rules I-A, I-B, IV, V, and VI.

I. For each T such that $\sum$ contains the declaration

$$\texttt{table}\ T(A1\ D1,\ \ldots,\ A_m\ D_m,\ \varphi_1,\ \ldots,\ \varphi_n)$$

define the basic ERD construct (entity type or relationship type) B_T as follows:

A. Choose the primary key of T to be a φ_i such that[4]

$$\varphi_i = \texttt{pathfd}\,(A_{i_1},\ldots,A_{i_k}) \rightarrow \texttt{self}$$

where $k \leq n$ and the indexes i on A denoting that they are part of constraint φ_i. The order between A_{i_j} in this constraint is irrelevant, so suppose they are already ordered such that all attributes participating in a FK constraint in T are put at the beginning, and all attributes that do not belong to any FK constraint are put at the end. Then let h be the index such that all attributes A_{i_j} with $j \leq h$ are also declared as FK, i.e., there is a constraint of the form

$$\varphi_j = \texttt{constraint}\,N_j\,\texttt{foreign key}\,(A_{i_j})\,\texttt{references}\,T_j$$

whereas for the remaining A_{i_j}, $h<j$ attributes in the PK such a FK constraint does not exist.

B. If h=0, i.e., there are no attributes in the PK that are also declared as an FK, then T represents a "strong relation" (*sensu* [30]). This step decides whether it is a strong entity type or a strong relationship type. If there exists a candidate key constraint

$$\varphi_{i'} = \texttt{pathfd}\,(A'_1,\ldots,A'_{k'}) \rightarrow \texttt{self}, i \neq i'$$

that defines another candidate key for T where all A'_j participate in a FK constraint in T, i.e., there exists

$$\texttt{constraint}\,N'_j\,\texttt{foreign key}\,(A'_j)\,\texttt{references}\,T_j$$

for all j, $1 \leq j \leq k'$ then set B_T to be a new relationship type with participation of all B_{T_j} basic constructs[5]. Otherwise, i.e., no such candidate key constraints exist, then let B_T be a new entity type.

C. If $h = k$, i.e., all attributes in PK are also FKs, then all A_j are included in foreign key constraints

$$\texttt{constraint}\,N_j\,\texttt{foreign key}\,(A_{i_j})\,\texttt{references}\,T_j$$

and T represents a "regular relation". Set B_T to be a new relationship in which those B_{T_j} participate that were reverse engineered from their respective T_j.

D. If $0<h<k$ then some PK attributes are included in FK constraints and some are not, so T represents a "weak relation". Then let B_T be a new entity type with partial key attributes determined by all the dangling key attributes A_j, $1 \leq j \leq h < k$.

Build the initial ERD with all B_T.

[4] There may be more than one of those pathfds specified, meaning that they are all candidate keys; in that case, the referring expressions machinery described in [9] resolves this to selecting the appropriate one automatically, which is assumed resolved here. Note that cycles are avoided by definition of the referring expression types (Definition 3 in [9]).

[5] For this process to be well-founded, a dependency graph with all T has to be built.

II. Assign all attributes A_{i_j} of PK φ_i where $h<j\leq k$ (i.e., those attributes of the PK that are not also FK) to B_T and declare them as members of the identifier (or partial identifier) for B_T.

III. Let A_l be any attribute that does not appear in the PK constraint φ_i, i.e., $l\neq i_j$ for all $1\leq j\leq k$. If A_l appears in a FK constraint in T

$$\texttt{constraint}\ N_l\ \texttt{foreign key}\ (A_l)\ \texttt{references}\ T_l$$

then add participation to B_{T_l} if it is a relationship type, or to a new binary relationship type if it is an entity type. In the latter case, include 1:n participation from B_T to B_{T_l}. If A_l does not appear in any FK constraint, then assign it to B_T as a regular attribute, ignoring its respective D_l.

IV. For each

$$\varphi_i = \texttt{isa}\ T_1$$

add a specialisation relationship between B_T and B_{T_1}.

V. For each

$$\varphi_i = \texttt{covered by}\ (T_1,\ldots,T_p)$$

used in conjunction with $\texttt{isa}\ T_i$, $(1\leq i\leq p$ and $p\geq 2)$, then declare the covering constraint over the corresponding isa in the ERD.

VI. For each

$$\varphi_i = \text{disjoint with}\ (T_1,\ldots,T_p)$$

used in conjunction with $\texttt{isa}\ T_i$, $(1\leq i\leq p$ and $p\geq 2)$, then declare the disjointness constraint over the corresponding isa in the ERD.

This transformation has a linear number of steps in the size of the relation declaration clauses. Step I.A selects the primary key, and partitions the set of attributes (no ordering is really necessary). A single scan of the table declaration clause is also enough for steps I.B, I.C, I.D, II and IV. Tagging non-PK attributes in step III can be done with the help of an appropriate data structure. A dictionary of isa-relationships can be built while executing step IV, which enables constraining each member in steps V and VI in constant time.

After this process, some information present in the ARM specification is not in the generated ERD, because EER does not support the features. In particular, all functional path dependencies other than those with consequent `self`, all datatype declarations for attributes, candidate key attributes, and all disjointness constraints that are not associated with any ISA relationship, are ignored. Thus, the transformation may not be information-preserving at the level of the model specification. However, this information may be attached

to the ERD as non-graphical constraints that later can be used in other tasks requiring a formal representation, such as reasoning or exporting to another language.

A possible reverse engineering of the ARM diagram of Example 2 is briefly illustrated now.

Example 3. *Let us continue from Example 2 and transform that ARM diagram into an EER diagram, whose outcome is included in Figure 4, having followed the procedure that is described in the previous section. For instance, let us take again the* `professor` *relation from Example 2. By rule I, we create an entity type* Professor *(Rule I-A selects the PK and relevant* pathfd (`pathfd` (`pnum`) -> `self`*), and notices that h=0 (rule I-B), so it is a strong relation), and add* pnum *as identifier to* Professor *(rule II). Add the other attributes not in a FK constraint as regular attributes to* Professor *and for the one as FK (*`department`*), create a new binary relationship to* Department *(rule III). This binary relationship will have to be named, and the rules applied to the* `department` *relation for the* Department *entity type in the EER diagram. Rules IV-VI do not apply, because there is no* `isa` *in the relation specification. This concludes processing the sequence of rules for this ARM relation.*

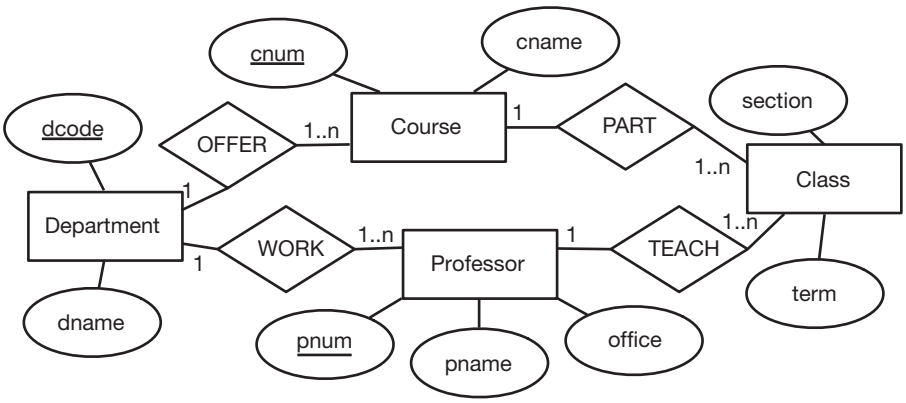

Figure 4. Possible EER diagram reconstructed from the ARM diagram of Example 2.

4.2 From EER Diagram to ARM

The algorithm from an ERD to an ARM follows the standard principles, such as FK on the n-side of a 1:n multiplicity on a relationship, but distinguishes itself from the staple algorithms on three main aspects: the `pathfds` with `self` for each entity type when generating an ARM relation, any other `pathfd` assertions, and, especially, the disjointness constraints also among relations for which there were entity types that are not in an isa hierarchy. These key differences *cf*. the well-known EER-to-RM algorithms can be observed in rules I-A, I-C, II-A, II-C, III, IV, V, and VI.

As notation convention, an entity type is denoted by E, its attributes as $A_1,\ldots,A_n$, and participating in relationships $R_1,\ldots R_m$ connecting to $E_1,\ldots,E_m$ in the EER diagram. The transformations assume the machinery of referring expressions to handle identification properly, as described in [9].

I. For each strong E with identifier composed by attributes $A_1,\ldots,A_l$ and regular attributes $A_{l+1},\ldots,A_n$, $1\leq l\leq n$,

A. Create in $\sum$ an ARM relation T_E with the declaration

$$T_E(\texttt{self OID}, A_1\ D,\ldots,A_n\ D, A_{n+1}\ \texttt{OID},\ldots,A_{n+m}\ \texttt{OID})$$

where D is `anyType` until the datatype has been set by the designer or retrieved from the reverse engineering step, and $A_{n+i}, 0\leq i\leq m$ are new attributes different from those existing in the ERD.

B. Add to T_E the FK constraints

$$\texttt{constraint}\ N_i\ \texttt{foreign key}\ (A_{n+i})\ \texttt{references}\ T_{E_i}$$

for all $i, 0\leq i\leq m$.

C. Add to T_E the PK constraint

$$\texttt{pathfd}\ (A_1,\ldots,A_l)\rightarrow \texttt{self}$$

II. For each weak E with partial identifier composed by attributes $A_1,\ldots, A_l$ and regular attributes $A_{l+1},\ldots,A_n$, where $1\leq l\leq n$, which is related through weak relations R_i[©], $1\leq i\leq k\leq m$.

A. Create an ARM relation T_E for E such that

$$T_E(\texttt{self OID}, A_1\ D,\ldots,A_n\ D, A_{n+1}\ \texttt{OID},\ldots,A_{n+m}\ \texttt{OID})$$

where D is anyType as before, and $A_{n+i}, 0\leq i\leq m$ are new attributes different from those existing in the ERD.

B. Add to T_E the FK constraints

$$\texttt{constraint}\ N_i\ \texttt{foreign key}\ (A_{n+i})\ \texttt{references}\ T_{E_i}$$

for all $i, 1\leq i\leq m$.

C. Add to T_E the PK constraint

$$\texttt{pathfd}\ (A_1,\ldots,A_l,A_{n+1},\ldots,A_{n+k})\rightarrow \texttt{self}$$

[©] Some of the R_i may not be strong, so a dependency graph must be constructed in order for this step to be well-founded.

III. For each subsumption, i.e., an E_1 ISA E in the ERD, take the "separation" option for transformation, so add to relation T_{E_1} in the ARM the constraint

$$\texttt{isa}\ T_E$$

which is shorthand for the pathfd constraint that the `self` of E_1 is an FK referencing the `self` in E.

IV. For each covering constraint declared in a hierarchy in the ERD, on $E_1,\ldots,E_n$ (with $n\geq 2$) that are entity subtypes of E, then add to T_E the constraint

$$\texttt{covered by}\ (T_{E_1},\ldots,T_{E_n})$$

V. For each disjointness declared in a hierarchy in the ERD, on $E_1,\ldots,E_n$ (with $n\geq 2$) that are entity subtypes of E, then add to each T_{E_i}, $1\leq i\leq n$ the constraint

$$\texttt{disjoint with}\ (T_{E_1},\ldots,T_{E_{i-1}},T_{E_{i+1}},\ldots,T_{E_n})$$

VI. Disjointness between ARM relations constructed from entity types that are not in a hierarchy in the ERD (and thus are implicitly disjoint) can be computed based on comparing their respective `pathfd` constraints that have as consequent `self` for each T_i, because each relation in the ARM resulting from an E_i (that is not in a hierarchy) of the ERD will have a different PK because they have different identifiers in the corresponding ERD.

With the exception of step V, all the previous steps of the transformation are linear in the size of the elements of the ERD since each deals with a specific type of element (strong entities, weak entities, isa's, etc.) and the output does not depend on this size. In step V, a single disjointness constraint generates as many constraints clauses as entities that participate in the constraint, so this step is quadratic. The resulting ARM after this process is a very weak representation of a database since several implementation details are missing, like attribute types, non-key functional dependencies and general disjointness constraints. The instantiation can be done automatically with external annotations (not present in the ERD), or manually with the help of an expert.

5. USING KNOWID BEYOND EER

The "extended ARM+SQLP" option for $D\bowtie K$, i.e., the KnowID architecture as shown in Figure 3, covers the knowledge-to-data pipeline, which was worked out for EER diagrams at the knowledge layer, a relational database as structured data, and SQLP as query language. In the light of it being a *generic* architecture, two central components require some clarifications to justify the claim to genericity, which is the query component with SQLP, and the formal representation beyond EER, i.e., addressing the other items in the box labelled Knowledge and Information Management in Figure 3. Considering the scope of this paper and that the theory and techniques that will be mentioned below have been introduced elsewhere already, we discuss these topics only from the architectural angle, not the details of the logics and algorithms.

5.1 SQLP over the ARM and EER Diagrams

Querying the data through the knowledge is an essential component of the knowledge-to-data pipeline, and therefore illustrated first. We first note that SQLP queries have been shown to be invariant under vertical partitioning, as that amounts to a lossless join decomposition [10]. Put differently, for an SQLP query over an ARM diagram, the query remains exactly the same irrespective of whether the graphical display of the model is denoted such that each class has attributes depicted "inside" the class or that they have (functional) relations where they are depicted as positioned alike another class (i.e., a graph-like graphical notation), because in both cases, formally, it amounts to the same formalization of the paths thanks to `self` and the `OID`s. We have tried to visuaze this in Figure 2 with two different graphical notations of ARM: on the left, the diagram has an RM look-and-feel, whereas on the right, this is flattened, where the attributes are shown as the (special) relationships that they are[7].

Because the EER-to-ARM, and vv., is transformation-based, the SQLP for ARM can propagate to its use with EER diagrams. The SQLP query does not use the relationship names explicitly, and hence, any naming assignment step for brand new names in case it was a reverse engineered EER diagram has no effect on the formulated query. The relevant vocabulary of EER's entity types, relationships and attributes, on the whole, remain the same with respect to the ARM, so the shape of the query can be the same. More precisely: the *written, textual*, and *version* of the query can be exactly the same[8] provided one takes into account that the names of the ERD relationships are ignored except for relationship names of relationships with m:n cardinality (which will become a relation in the ARM diagram). Thus, cognisant of the fact that normally CWA is assumed with EER already, this can indeed be maintained in KnowID thanks to a transformation-based approach and therewith using the option to remain with SQLP. Let us illustrate that in the following example.

Example 4. *We continue with our running example from Example 3, and the ERD of Figure 4. A sample query could be, e.g., "*Find the distinct names of all the professors who work at the CS department that have taught a class of a course offered by another department.*". This can be likewise executed over an ARM or EER diagram, by availing of the link from the ARM-to-EER transformation and constructing paths "from-the-n-to-the-1-direction-into-an-attribute". Moreover, this should be easier when queried over the ERD, because the names of the relationships that are present in the EER diagram loosely map onto verbs in the information request. In this case, as follows:*

"Find the distinct names of all the professors who work at the CS department that have taught [teach] a class [part] of a course offered by another department."

This is not the case in an ARM-as-RM-lookalike diagram. Alternatively, in a graphical interface, the path can be selected that includes the relationships in the graphics, which makes it visually clearer that there is that path. Such choices are left to an interface design stage.

[7] The question of what would constitute the best interface for user interaction is a separate line of research; we merely want to indicate that it is possible to change the display.

[8] The display of a visual query may look different for an end-user depending on the chosen user interface.

5.2 Flexibility in the KnowID Input

We now look into the Knowledge and Information Management box in Figure 3, with its four core sub-processes drawn above that, which depend on the precise input of "conceptual data model or application ontology *C*". For instance, the input may be a conceptual data model in UML class diagram notation or an Object-Role Modeling (ORM) diagram. Also, there are several formalizations of EER in various logics, invariably using the OWA for automated reasoning, rather than the CWA common for the database and conceptual modeling setting, which has to be addressed.

Those four steps are briefly explained as follows, and elaborated on afterward:

1. If *C* is not in EER, then convert it into EER, using a mapping or metamodel-driven approach; see, e.g., [31, 32] and references therein.
2. If the EER diagram was not formalized yet, then use one of the logic-based reconstructions to formalize it (elaborated on below).
3. Taking the formalization as input, compute the inferences (deductions); if there are undesirable deductions or unsatisfiable classes, revise the model and classify it again; repeat the process until the deductions, if any, are acceptable.
4. Materialize the deductions (if any), i.e., modify the EER diagram by adding the acceptable deductions; one may wish to double-check the modifications by classifying again, which should return an empty list of deductions.

The model resulting from completing step 4 is the one that will be transformed into an ARM and used for querying the data.

To determine a suitable logic for either step 2 or for having the conceptual model formalized in the first place, we note it ought to be able to formalize at least the following modeling language features in a suitable logic, because they can be dealt with in the ARM and/or RM: entity type (weak and strong), *n*-ary relationship, attribute, basic cardinality constraints (0..n, 0..1, 1, 1..n) and identifier, entity type subsumption, disjointness, and covering/total. The remaining EER features, such as higher number restrictions (e.g., 2..4) and multivalued attributes, are generally transferred directly into the physical schema only (if at all), and this "feature gap" is thus not an issue specific to this architecture with the ARM, so therefore acceptable to set aside. A good fit regarding the desired features list is $\mathcal{DLR}_{\text{ifd}}$ [33] of the Description Logics family, but it is ExpTime-complete in subsumption reasoning. A disadvantage of $\mathcal{DLR}_{\text{ifd}}$ is that does not have tools, and therefore users are likely to turn to OWL 2 DL [15] or a fragment thereof [34] and one of the DL reasoners thanks to their ample tooling support, despite the feature mismatch (among others, no *n*-aries, no multi-attribute identifiers). KnowID as approach permits either. For better performance, one could drop covering/total and use $\mathcal{CFDI}^{\forall-}_{nc}$ [24], which is in PTime. Others, notably those in the computationally better behaved DL-Lite family [35], have logic-based reconstructions of EER (e.g., [35]), but have even fewer features at a modeler's disposal. Because the classification reasoning task is separated from querying and can be done "offline" (i.e., not at runtime), there is no need to opt for the low expressiveness.

Regarding step 4, the notion of materializing deductions is not new and was a feature in an earlier version of the Protégé ontology development environment [36]; e.g., if A ISA B is deduced but not asserted in the EER diagram, one adds that explicitly, even though it is "redundant" because implicitly present. The advantage of materializing all the deductions at the knowledge layer, is that then that reasoning service is not needed anymore at runtime, saving processing time. We deem this an acceptable approach, since a conceptual model or ontology does not change very frequently. Regarding the deductions, in OWA they are not the same as in CWA, but in the research on reasoning over conceptual models, this difference is widely ignored (i.e., *de facto* accepted that there is no good alternative). Here, we follow that same approach for this not-at-runtime step. Note that this does not affect the CWA behavior that users expect during interaction with the system, because it is effectively "closed off" with this fourth step.

5.3 Flexibility in KnowID's Data Layer

The KnowID architecture presented in Figure 3 has a box labelled "Database schema(s) *S*" and only one RM listed. This is because, practically, there may be an SQL federator at the database schema layer to provide a global schema at the implementation level, as is possible with OBDA systems as well. Such a federator may deliver an RM as global view, which is, for both the basic ARM+SQLP, and thus also the here proposed KnowID, the starting point for the principal novelties that make up KnowID. Therefore, the data storage component of KnowID as architecture, is as flexible as current theories, techniques, and tools for relational database systems and any other data storage that can be queried directly or indirectly with SQL.

Secondly, the ARM+SQLP component within KnowID in Figure 3 includes a Data completion step that uses the data and an ARM (which also could be the formalized EER diagram that uses constraints supported in the EER-to-ARM conversion). Data completion, informally, means that the deductions are computed first and then materialized in the database. Let us return to Example 1 to illustrate this process and how it operates within KnowID.

Example 5. *To recall, the database stores* `IndividualCustomer` ⟨`Joanne`⟩ *and the conceptual data model or ontology contains* IndividualCustomer ⊑ Customer. *The data completion process fires an algorithm that checks the data against the knowledge. In this case, it is alike:*

1. *Observation: there is an instance of* IndividualCustomer*, being* {`Joanne`}*, which is declared in the database table* `IndividualCustomer`.
2. *Action: use the schema-to-RM-to-ARM transformation logs to look up what the corresponding relation in the ARM model says about* `IndividualCustomer`.
3. *Observation: it is a* Customer*, as can be read in the constraint* `isa Customer` *that is declared in the* `IndividualCustomer` *relation.*
4. *Action: deduce that the instances of* IndividualCustomer *are also instances of* Customer*, availing of the semantics of the* `isa` *constraint in the ARM specification.*
5. *Action: use the ARM-to-RM-to-schema transformation log to then append the* `Customer` *table (originally generated from* Customer*) with* `Customer` ⟨`Joanne`⟩.

Now the answer to "list all customers" will be, indeed, {Joanne}*, as one would expect from human understanding.*

There are trade-offs between this data completion approach and query rewriting on-the-fly that is typically incorporated in the *K*⇔*D* architecture. Determining which one is the best option for a particular application, depends on the frequency of updates, whether any optimizations are used in the data completion stage (e.g., incrementally [37]), and the tolerance for errors due to out-of-dateness of the data. If there are (near) continuous updates, data completion tends to be costly because it keeps re-computing and storing the implied assertions, but if that is not continuous or an incremental approach to data completion can be taken [37], it is favored over costly query rewriting, which is exponential in the size of the query [38] compared to data growing only polynomially in the size of the data for the queries [39], although this polynomial blowup in the data may still be unacceptable for large data sets. Optimization of rematerialization after updates in the conceptual model is another aspect that must be addressed in future works.

Finally, one could take the SQL result and use R2RML [40] or similar [41] to generate RDF triples from it so as to integrate it with a Web-based knowledge graph, which is already possible with the *K*⇔*D* and *K*@*D* approaches.

6. EVALUATION

We have conducted evaluations of KnowID in two different ways:

- Proof-of-concept implementation of the transformation rules;
- Comparison against OBDA (*K*⇔*D*) as main contender, as well as *K*@*D* and *D*@*K*; which are dealt with consecutively in the remainder of this section.

The aim of the evaluation with a proof-of-concept tool was to ascertain experimentally whether the transformation rules were indeed correct, using test models in EER and ARM to cover all cases. A preliminary flexible tool was designed and implemented, which takes EER and ARM diagrams serialized in JSON, transforms them at the back-end into an array, applies the relevant transformation rules, and returns an ARM or EER diagram and a log file with the record of the transformation (see Figure 5) [42]; the code, some examples, and a few screenshots are available at http://www.meteck.org/KnowID.html. In total, 83 tests were run with the latest version. They were designed such that they would be covering all rules as well as unsupported features to verify it would reject those in the transformation. It showed that the transformation rules defined in Section 4 are indeed concise and correct. It also made clear that some aspects of the transformation are easier algorithmically when stated more explicitly in the serialization of the models. A notable example of that is the `isa` constraint (rule III in Section 4.2) when taken in the context of computing disjointness among the ARM relations (rule VI in Section 4.2): a brute-force computation among all `pathfd` constraints to `self` with a simple foreign key check is easier to implement than unpacking the `isa` shorthand notation.

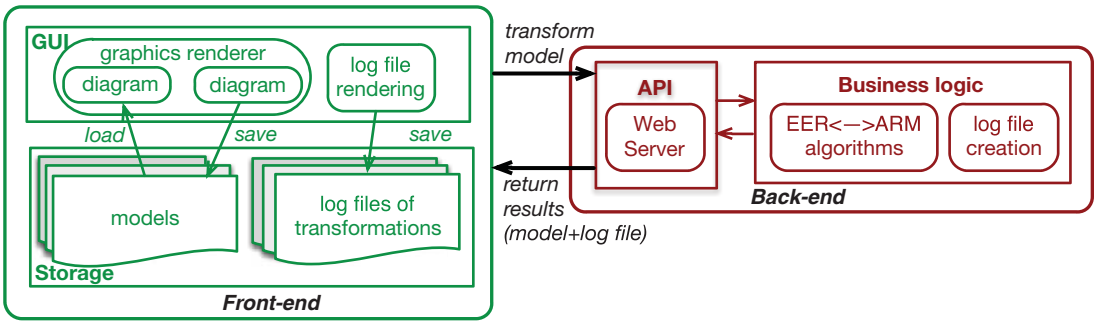

Figure 5. Overview of the KnowID transformation tool architecture.

We investigated the possibility of an experimental evaluation of KnowID against an instance of *K⇔D* (OBDA), such as *Ontop*, on performance of one or more tasks. Aside from any implementation optimizations, from the respective architectures, the variables determining overall complexity (recall Table 1), and their strengths, it is easy to construct a set-up that is guaranteed to have one or the other win on performance. Therefore, the aim of the comparison evaluation presented here is to highlight those crucial factors for a knowledge-to-data system that will make one or the other win when actual instances are compared, i.e., looking at practical consequences of the theoretical generic differences. The three key factors are:

1. *Stability of the data:* Do they (i) continuously change a lot throughout the database or (ii) intermittently, rarely, or append-only?
2. *Stability of the schemas:* Do they (i) remain unchanged once the system is set up or (ii) will they have to change based on changing business needs and usage optimizations?
3. *Type of queries posed over the data:* Are they (i) at most (unions of) conjunctive queries (UCQs) or (ii) also other types of SQL queries (with or without paths)?

If the data change continuously, the schemas remain stable, and one needs at most UCQs, then OBDA will be the winner, whereas for intermittent, rarely, or append-only data updates and UCQs+any other SQL (or SQLP) query, KnowID will win (irrespective of schema stability). The reasons for this outcome are as follows. With many database updates, KnowID will take up many resources to keep recomputing the data completion, which is not an issue for OBDA because it does those inferences on-the-fly for that query only. While those on-the-fly inferences are computationally costly, there is a tipping point where it outperforms the data completion approach in praxis (recall Section 5.3). For the type of queries, it would be easy to construct a list of user queries for evaluation that includes queries that are not UCQs, on which OBDA thus fails outright and KnowID wins since it supports full SQL and thus certainly will obtain a 100% pass rate on the user queries. Finally, creating the mapping layer in OBDA is resource-intensive with a human-in-the-loop, and hence, time consuming to maintain when there are changes to either the ontology or the database schema. Conversely, the transformations in KnowID are computed automatically and simply can be recomputed on-the-fly in the face of changes, be they change conceptually or in the implementation, since the transformation algorithms are specified in both directions.

Configuring experiments to pit KnowID against the other two approaches mentioned in Section 2.1, *K@D* with OWL and *D@K* with OntoMIND, also would be easy to make one or the other win, but for different reasons. It is well-known that *K@D* with only an OWL file already shows noticeable performance degradation when there 10s or 100s of individuals declared in the ontology (depending on the axioms), which is precisely the reason for devising the other architectures; or: this is no real contest. The *D@K* implementations typically have no reasoning component, so a query that requires it will lose against KnowID due to returning fewer tuples than logically implied (recall {} *vs*. {`Joanne`} from Example 1). The one *D@K* system that has some form of reasoning with triggers and stored procedures, OntoMIND [7], does not support *n*-aries and cardinalities in its ontology language (*DL-Lite*$_{\mathcal{R},\sqcap}$), and also uses data completion. Since *DL-Lite*$_{\mathcal{R},\sqcap}$ is a fragment of our language for EER, and hence, there is less to compute both in the conceptual model and in the data completion process, it should be faster but also retrieve fewer tuples than it ought to on carefully constructed queries.

Overall, KnowID as an instantiation of *D⋈K* evaluates positively against *K@D* and *D@K* in the light of the knowledge-to-data pipeline in any case, and, depending on the characteristics of the prospective implementation, positively against *K⇔D* as well.

7. CONCLUSIONS AND FUTURE WORK

The paper presented a novel "knowledge to data" pipeline, called KnowID, that combines new transformation rules between EER and the Abstract Relational Model with recently proposed components for the data, information, and query components. KnowID's main distinctive features are that runtime use can avail of the closed world assumption with an RDBMS, relational and abstract relational model, and EER as declarative specifications and full SQL with path queries for expressive compact queries formulated over the knowledge.

We are currently investigating the solution space of implementing the transformation rules as a first step toward realizing KnowID as a usable and scalable software system. With the rules having been verified working in the implementation, we are considering multiple usage scenarios, such as the graphical interfaces for both the conceptual model or ontology and an ARM model, usability and human readability of the transformation log in anticipation of knowledge-based querying with SQLP, and compatibility with other tooling infrastructures. The current code and examples are, and anticipated updates will become, available from http://www.meteck.org/KnowID.html.

AUTHOR CONTRIBUTIONS

P.R. Fillottrani (prf@cs.uns.edu.ar) and C.M. Keet (mkeet@cs.uct.ac.za) jointly developed the theory. C.M. Keet coordinated the writing of the manuscript and devised the examples. Both authors have made meaningful and valuable contributions in revising and proofreading the resulting manuscript.

ACKNOWLEDGEMENTS

We thank David Toman for feedback on an earlier draft. We also thank the 20 students of the 7 capstone projects for exploring the solution space in implementing the transformation rules.

REFERENCES

[1] T. Catarci & G. Santucci. Query by diagram: A graphical environment for querying databases. ACMSIGMOD Record 23(2)(1994), 515. doi:10.1145/191843.191976.

[2] A.C. Bloesch & T.A. Halpin. ConQuer: A conceptual query language. In: Proceedings of ER'96: 15th International Conference on Conceptual Modeling, 1996, pp. 121–133. doi:10.1016/0169-023X(95)00005-D.

[3] G. Xiao, L. Ding, B. Cogrel & D. Calvanese. Virtual knowledge graphs: An overview of systems and use cases. Data Intelligence 1 (2019), 201–223. doi:10.1162/dint a 00011.

[4] M. Stonebraker & I.F. Ilyas. Data integration: The current status and the way forward. IEEE Data Engineering 41(2)(2018), 3–9. Available at: http://sites.computer.org/debull/A18june/p3.pdf.

[5] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro & G. Xiao. Ontop: Answering SPARQL queries over relational databases. Semantic Web Journal 8(3)(2017), 471–487. doi:10.3233/SW-160217.

[6] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini & R. Rosati. Linking data to ontologies. In: Spaccapietra S. (eds.) Journal on Data Semantics X, 2008, pp. 133–173. doi:10.1007/978-3-540-77688-8_5.

[7] L. Al-Jadir, C. Parent & S. Spaccapietra. Reasoning with large ontologies stored in relational databases: The Onto-MinD approach. Data & Knowledge Engineering 69(2010), 1158–1180. doi:10.1016/j.datak.2010.07.006.

[8] F. Zhang, Z.M. Ma & W. Li. Storing OWL ontologies in object-oriented databases. Knowledge-Based Systems 76(2015), 240–255. doi:10.1016/j.knosys.2014.12.020.

[9] A. Borgida, D. Toman & G.E. Weddell. On referring expressions in information systems derived from conceptual modeling. In: Proceedings of ER'16, 2016, pp. 183–197. doi:10.1007/978-3-319-46397-1_14.

[10] W. Ma, C.M. Keet, W. Oldford, D. Toman & G. Weddell. The utility of the abstract relational model and attribute paths in SQL. In: C. Faron Zucker, C. Ghidini, A. Napoli & Y. Toussaint (eds.) Proceedings of the 21st International Conference on Knowledge Engineering and Knowledge Management (EKAW'18)), 2018, pp. 195–211. doi:10.1007/978-3-030-03667-6_13.

[11] M. Junkkari, J. Vainio, K. Iltanenan, P. Arvola, H. Kari & J. Kekäläinen. Path expressions in SQL: A user study on query formulation. Journal of Database Management 22(3)(2016), 22. doi:10.4018/JDM.2016070101.

[12] C.M. Keet. An introduction to ontology engineering. Available at: http://open.uct.ac.za/bitstream/handle/11427/28312/OEbook.pdf?sequence=1&isAllowed=y.

[13] P.R. Fillottrani & C.M. Keet. Dimensions affecting representation styles in ontologies. In: The 1st Iberoamerican Conference on Knowledge Graphs and Semantic Web (KGSWC'19), 2019, pp. 186–200. doi:10.1007/978-3-030-21395-4_14.

[14] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P.F. Patel-Schneider (eds.). The Description Logics Handbook – Theory and Applications. 2nd Edition. Cambridge: Cambridge University Press, 2010. isbn: 9780521150118.

[15] B. Motik, P.F. Patel-Schneider & B. Parsia. OWL 2 web ontology language structural specification and functional style syntax, W3c recommendation, W3C. Available at: http://www.w3.org/TR/owl2-syntax/.

[16] D. Toman & G.E. Weddell. Fundamentals of physical design and query compilation, synthesis lectures on data management. Williston, VT: Morgan & Claypool Publishers, 2011. doi:10.2200/S00363ED1V01Y-201105DTM018.

[17] D. Calvanese, T.E. Kalayci, M. Montali, A. Santoso & W. van der Aalst. Conceptual schema transformation in ontology-based data access. In: C.F. Zucker, C. Ghidini, A. Napoli & Y. Toussaint (eds.) Proceedings of the 21st International Conference on Knowledge Engineering and Knowledge Management, 2018, pp. 50–67. doi:10.1007/978-3-030-03667-6_4.

[18] E. Botoeva, D. Calvanese, B. Cogrel, J. Corman & G. Xiao. A generalized framework for ontology-based data access. In: C. Ghidini, B. Magnini, A. Passerini & P. Traverso (eds.) Proceedings of AI*IA'18, 2018, pp. 166–180. doi:10.1007/978-3-030-03840-3_13.

[19] E. Kharlamov, D. Hovland, M.G. Skaeveland, D. Bilidas, E. Jiménez-Ruiz, G. Xiao ... & A. Waaler. Ontology based data access in Statoil. Web Semantics: Science, Services and Agents on the World Wide Web 44 (2017), 3–36. doi:10.1016/j.websem.2017.05.005.

[20] A. Artale, D. Calvanese, R. Kontchakov & M. Zakharyaschev. DL-Lite without the unique name assumption. In: Proceedings of the 22nd International Workshop on Description Logic (DL 2009), 2009, pp. 1–13. Available at: http://ceur-ws.org/Vol-477/paper_11.pdf.

[21] N.E. Fuchs, K. Kaljurand & T. Kuhn. Discourse representation structures for ACE 6.6, Technical Report, ifi-2010.0010, Department of Informatics, University of Zurich, Switzerland, 2010. Available at: http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_66.pdf.

[22] D. Calvanese, C.M. Keet, W. Nutt, M. Rodríguez-Muro & G. Stefanoni. Web-based graphical querying of databases through an ontology: the WONDER system. In: S.Y. Shin, S. Ossowski, M. Schumacher, M.J. Palakal & C.C. Hung (eds.) Proceedings of ACM Symposium on Applied Computing (ACM SAC'10), 2010, pp. 1389–1396. doi:10.1145/1774088.1774384.

[23] A. Soylu, E. Kharlamov, D. Zheleznyakov, E.J. Ruiz, M. Giese, M. Skjaeveland, D. Hovland ... & I. Horrocks. Optiquevqs: A visual query system over ontologies for industry. Semantic Web 9(5)(2018), 627–660. doi:10.3233/SW-180293.

[24] D. Toman & G.E. Weddell. On adding inverse features to the description logic CFD8nc. In: PRICAI 2014: Trends in Artificial Intelligence - 13th Pacific Rim International Conference on Artificial Intelligence, 2014, pp. 587–599. doi:10.1007/978-3-319-13560-1_47.

[25] J.S. Jacques, D. Toman & G.E. Weddell. Object-relational queries over CFDInc knowledge bases: OBDA for the SQL-Literate. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2016, pp. 1258–1264. Available at: http://ceur-ws.org/Vol-1577/paper_10.pdf.

[26] T. Halpin & T. Morgan. Information modeling and relational databases. 2nd Edition. San Francisco, CA: Morgan Kaufmann, 2008. isbn: 9780123735683.

[27] P.R. Fillottrani, C.M. Keet & D. Toman. Polynomial encoding of ORM conceptual models in $\mathcal{CFDI}_{nc}^{\forall-}$. In: D. Calvanese & B. Konev (eds.) Proceedings of the 28th International Workshop on Description Logics (DL'15), 2015, pp. 401–414. Available at: http://ceur-ws.org/Vol-1350/paper-50.pdf.

[28] C.M. Keet & P.R. Fillottrani. An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2. Data & Knowledge Engineering 98(2015), 30–53. doi: 0.1016/j.datak.2015.07.004.

[29] P.R. Fillottrani & C.M. Keet. Evidence-based languages for conceptual data modeling profiles. In: T. Morzy et al. (eds.) The 19th Conference on Advances in Databases and Information Systems (ADBIS'15), 2015, pp. 215–229. doi:10.1007/978-3-319-23135-8_15.

[30] R.H. Chiang, T.M. Barron & V.C. Storey. Reverse engineering of relational databases: Extraction of an EER model from a relational database. Data & Knowledge Engineering 12(2)(1994), 107 – 142. doi:10.1016/0169-023X(94)90011-6.

[31] P.R. Fillottrani & C.M. Keet. Conceptual model interoperability: A metamodel-driven approach. In: A. Bikakis et al. (eds.) Proceedings of the 8th International Web Rule Symposium (RuleML'14), 2014, pp. 52–66. doi:10.1007/978-3-319-09870-8_4.

[32] Z.C. Khan, C.M. Keet, P.R. Fillottrani & K. Cenci. Experimentally motivated transformations for intermodal links between conceptual models. In: J. Pokorný et al. (eds.) The 20th Conference on Advances in Databases and Information Systems (ADBIS'16), 2016, pp. 104–118. doi:10.1007/978-3-319-44039-2_8.

[33] D. Calvanese, G. De Giacomo & M. Lenzerini. Identification constraints and functional dependencies in description logics. In: B. Nebel (ed.) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), 2001, pp. 155–160.

[34] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue & C. Lutz. OWL 2 Web Ontology Language Profiles, W3C recommendation, W3C (27 Oct. 2009). Available at: http://www.w3.org/TR/owl2-profiles/.

[35] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov & M. Zakharyaschev. Reasoning over extended ER models. In: C. Parent et al. (eds.) Proceedings of the 26th International Conference on Conceptual Modeling (ER'07), 2007, pp. 277–292. doi:10.1007/978-3-540-75563-0_20.

[36] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson ... & S.W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. International Journal of Human-Computer Studies 58 (1) (2003), 89–123. doi:10.1016/S1071-5819(02)00127-1.

[37] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu & J. Banerjee. Rdfox: A highlyscalable RDF store. In: M. Arenas (ed.) Proceedings of the International Semantic Web Conference (ISWC'15), 2015, pp. 3–20. doi:10.1007/978-3-319-25010-6_1.

[38] G. Gottlob, S. Kikot, R. Kontchakov, V.V. Podolskii, T. Schwentick & M. Zakharyaschev. The price of query rewriting in ontology-based data access. Artificial Intelligence 213 (2014), 42–59. doi:10.1016/j.artint.2014.04.004.

[39] C. Lutz, D. Toman & F. Wolter. Conjunctive query answering in the description logic EL using a relational database system. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'09), 2009, pp. 2070–2075. Available at: http://ijcai.org/Proceedings/09/Papers/341.pdf.

[40] S. Das, S. Sundara & R. Cyganiak. R2RML: RDB to RDF mapping language. Available at: https://www.w3.org/TR/r2rml/.

[41] Ó. Corcho, F. Priyatna & D. Chaves-Fraga. Towards a new generation of ontology based data access. Semantic Web 11 (1) (2020), 153–160. doi:10.3233/SW-190384.

[42] P.R. Fillottrani, S. Jamieson & C.M. Keet. Connecting knowledge to data through transformations in KnowID: system description. Submitted to a journal, 2019.

AUTHOR BIOGRAPHY

Pablo Rubén Fillottrani is a Professor with the Department of Computer Science and Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina, and independent researcher at Comisión de Investigaciones Cientficas de la Provincia de Buenos Aires. He is head of LISSI, Software Engineering and Information Systems R&D Lab. His research interests are in ontology engineering, semantic Web, knowledge representation and information integration with applications in digital government and software engineering. Pablo obtained his PhD at Universidad Nacional del Sur in 2001.
ORCID: 0000-0003-0906-867X

C. Maria Keet is an Associate Professor with the Department of Computer Science, University of Cape Town, South Africa. Her research interests are in knowledge engineering, including ontology engineering, natural language generation, and ontology-driven conceptual modeling, which have resulted in over 100 publications. She wrote a textbook on ontology engineering. Maria obtained her PhD at the KRDB Research Centre, Free University of Bozen-Bolzano, Italy, in 2008. She also has worked as systems engineer in the IT industry for three and half years.
ORCID: 0000-0002-8281-0853